Module: COMP1752 Object-Oriented Programming	**Coursework**
Contribution: 100% of course	**ZIP file required, containing PDF report & PyCharm project**
Module Leader: Dr Nageena Frost	**Due date:**
This coursework should take an average student who is up-to-date with tutorial work approximately 50 hours	

Learning Outcomes:

1. Recognise and apply principal features of object-oriented design such as abstraction, encapsulation, inheritance and polymorphism.
2. Design non-trivial programmes with a view to flexibility and reuse using appropriate design methods.
3. Code, test and evaluate small software systems to conform to a specification.

Plagiarism is presenting somebody else's work as your own. It includes copying information directly from the Web or books without referencing the material; submitting joint coursework as an individual effort; copying another student's coursework; stealing or buying coursework from someone else and submitting it as your own work. Suspected plagiarism will be investigated and if found to have occurred will be dealt with according to the procedures set down by the University.

All material copied or amended from any source (e.g. internet, books) must be referenced correctly according to the reference style you are using.

Your work will be submitted for electronic plagiarism checking. Any attempt to bypass our plagiarism detection systems will be treated as a severe Assessment Offence.

Coursework Submission Requirements

- An electronic copy of your work for this coursework should be fully uploaded by Deadline Date.
- The last version you upload will be the one that is marked.
- For this coursework you must submit a **single zip file** containing your report, in an **Acrobat PDF document**, and supporting code in PyCharm project. In general, any text in the document must not be an image (i.e. must not be scanned) and would normally be generated from other documents (e.g. MS Office using "Save As ... PDF").
- There are limits on the file size (currently 2Gb).
- Make sure that any files you upload are virus-free and not protected by a password or corrupted otherwise they will be treated as null submissions.
- Feedback on your work will be available from Moodle. The grade will be made available in the portal.
- You must NOT submit a paper copy of this coursework.
- All coursework must be submitted as above.

The University website has details of the current Coursework Regulations, including details of penalties for late submission, procedures for Extenuating Circumstances, and penalties for Assessment Offences.
See https://www.gre.ac.uk/student-services/regulations-and-policies for details.

Coursework Specification – Jukebox Simulation

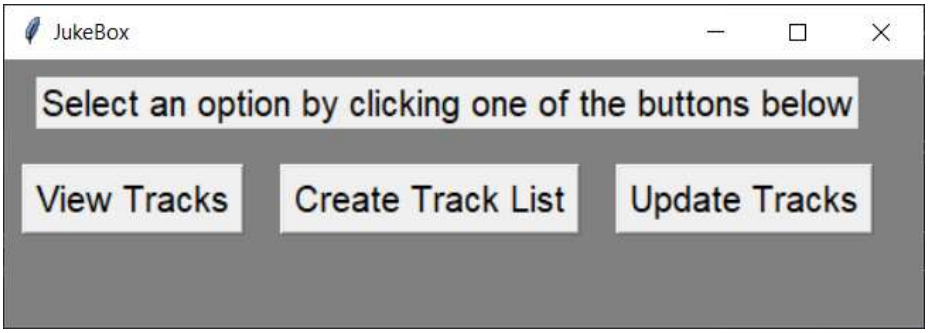
You are to produce a simulation of a jukebox app to play music (something like a very basic version of Windows Media Player).

You should start with the template code in a PyCharm project called JukeBox, which you can download from Moodle.

The template project contains the following files:

track_player.py:

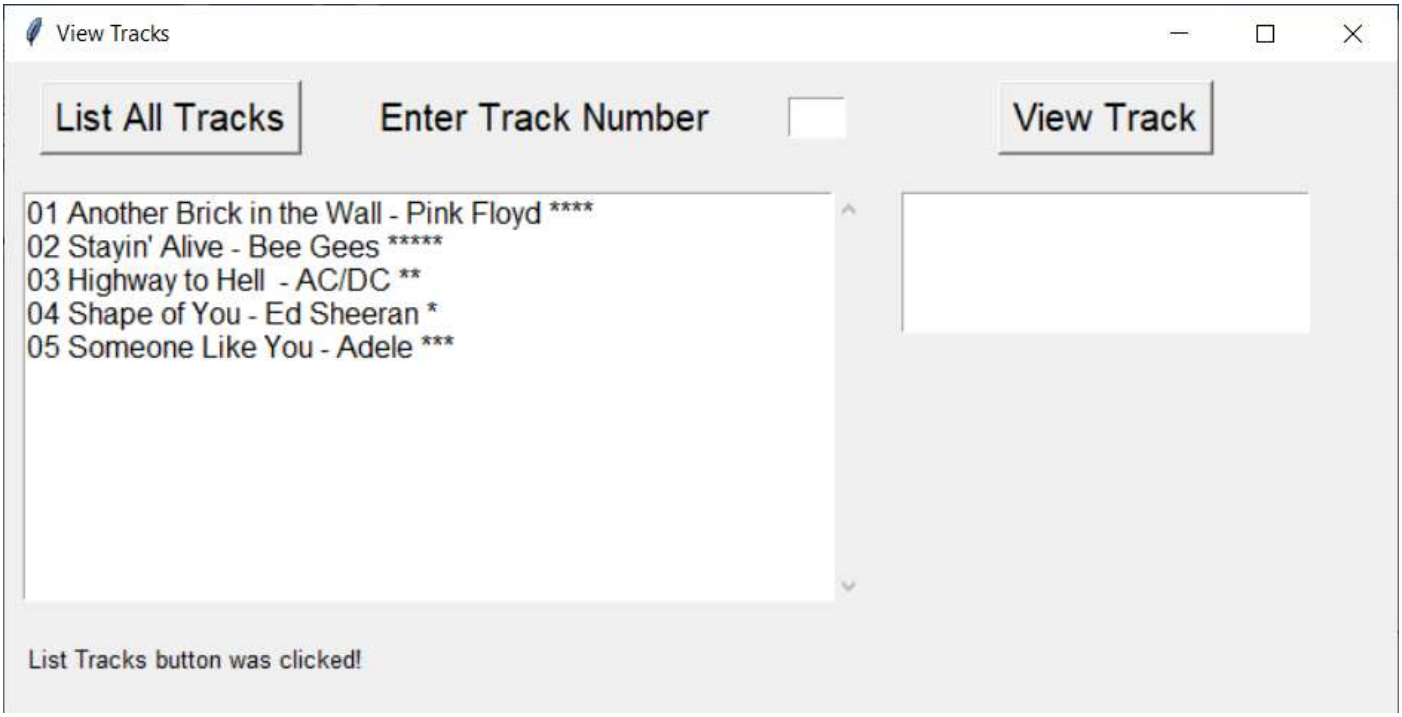
This is what the jukebox player GUI looks like when run:



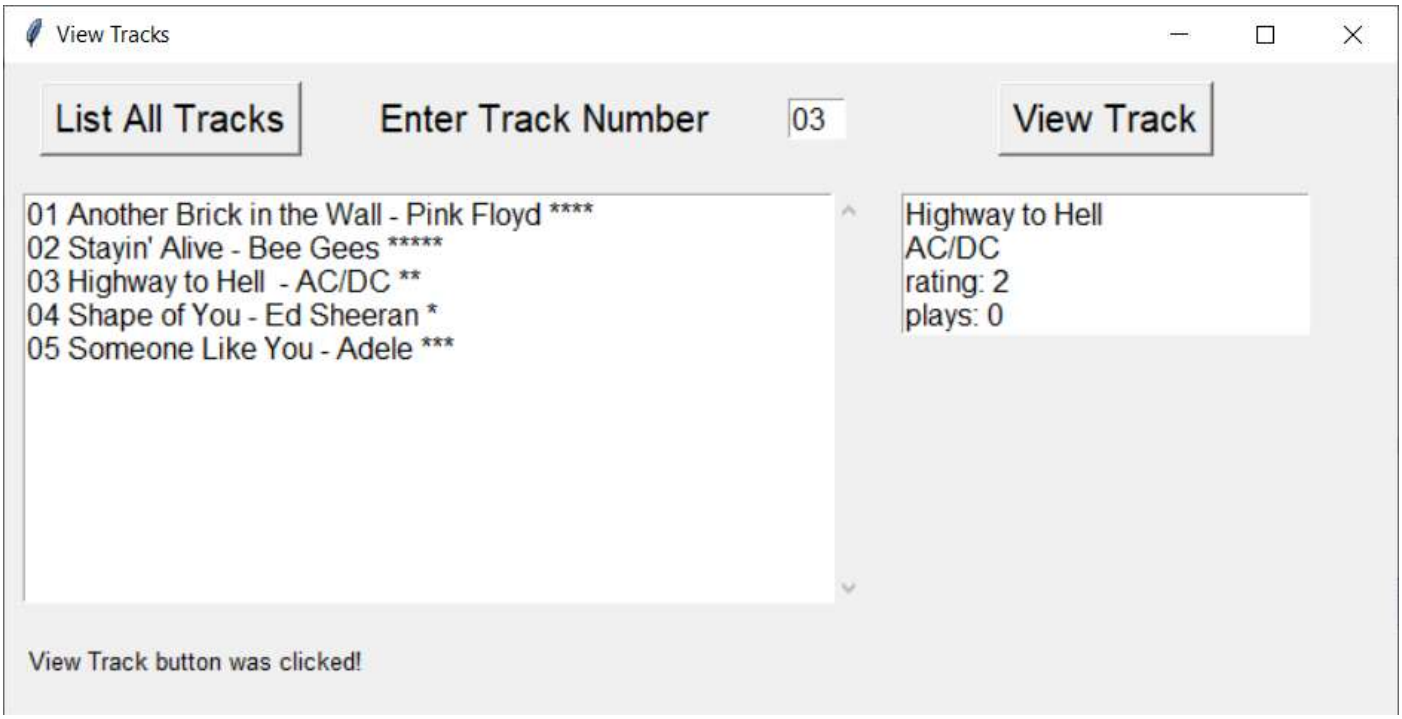
The **Create Track List** and **Update Tracks** buttons do nothing at this stage.

view_tracks.py:

When the user clicks the **View Tracks** button the following GUI appears (with the JukeBox GUI still displayed on screen). The GUI shows an identifying track number for each track, the name of the track, the name of the artist followed by rating shown as stars.



If the user enters a valid track number and clicks the **View Track** button, details of the track are displayed. If the track number is invalid, an error message is displayed instead.



track_library.py:

You will also be supplied with the **track_library.py** file which holds the list of tracks as a hard-coded database of LibraryItem objects. This provides number of functions which you may use:

- `list_all()`
- `get_name()`
- `get_artist()`
- `get_rating()`
- `set_rating()`
- `get_play_count()`
- `increment_play_count()`

For instance the `get_name()` function is used by **view_tracks.py** to obtain the name of the item with the given track number. It returns `None` if there is no such item in the library.

font_manager.py:

Finally, you will be supplied with **font_manager.py** file which just sets font styles and sizes for all the GUIs. You do not need to modify this, but you may, if you wish to customise the appearance.

Basic Functionality

Using **view_tracks.py** as an example you are to write two GUIs, **create_track_list.py** and **update_tracks.py**, with the following functionality.

create_track_list.py should allow the user to:

- Enter a track number and click a button to add that track to a playlist.
- If the track number is valid, the track name should be added to the list and all the names in the list should be displayed in a text area, otherwise a suitable error message should be displayed.
- Click a button to play the playlist – since this is just a simulation no tracks will be played, but each track on

the playlist should have its play count value incremented by 1.
- Click a button to reset the playlist and clear the text area.

update_tracks.py should allow the user to enter a track number and a new rating. If the track number is valid a message showing the track name, the new rating and the play count should be displayed, otherwise a suitable error message should be displayed.

Coursework Stages

There are a number of stages to the development and not all students will manage all the stages. Each stage will be awarded the marks up to the maximum allocated. However, you cannot get marks for a particular stage unless you have made a reasonable attempt at **all** of the previous stages. You may also lose marks if your code contains runtime errors or your report is missing one or more sections.

Stage 1 Basic understanding (max 10% including report)

view_tracks.py should be rewritten with comments explaining exactly how all the code works, e.g.

```
def set_text(text_area, content):    # inserts content into the text_area
    text_area.delete("1.0", tk.END)  # first the existing content is deleted
    text_area.insert(1.0, content)   # then the new content is inserted
```

Stage 2 Outline implementation (max 10% including report)

create_track_list.py and **update_tracks.py** should be designed and the code written in prototype form WITHOUT functionality (just the GUI appearance).

Stage 3 A basic working version (max 30% including report)

Implement the system whose GUI you designed in stage 2.

Stage 4 Testing and validation (max 20% including report)

Adapt the code to perform validation, i.e. checking for bad input such as a value of 'four' rather than '04'.

Design *unit testing* of the LibraryItem class using PyTest.

Finally design a series of tests that check the functionality of your GUI. List these tests in a table giving:

- sample input, if any (e.g. a track number)
- sample actions (e.g. pressing a particular button)
- expected output
- actual output, if different – i.e. if the test has failed

Stage 5 Innovations (max 20% including report) , Reflections and Future Developments (max 10%)

Adapt the code to include at least two innovations. Here are some examples but you can also devise your own:

- Display a picture to accompany each track when the user checks it.
- Provide a search facility, for example to search for tracks/artists by name.
- Redesign the system so it uses a single GUI.
- Provide a filter option, for example to view all the tracks of a selected artist.
- Store the track library data externally in a csv file which is updated every time the data changes.
- Enhance the play list facility, for example to allow playlists to be saved externally and modified later.
- Implement additional LibraryItem subclasses (e.g. LibraryItemAlbum for a track in an album) which inherit from LibraryItem and also contain additional attributes and/or methods.

Interim Deliverables

There are two interim deliverables.

Both interim deliverables should be demonstrated to your tutor.

Interim DELIVERABLE A (Stages 1 & 2)

For Deliverable A you must present your commented code for stage 1 and the outline implementation for stage 2.

Deliverable A should be presented a week after the coursework is released.

Interim DELIVERABLE B (Stage 3)

For Deliverable B you must present a basic working version of the code.

Deliverable B should be presented two weeks before the coursework is due.

There are no marks for interim deliverables. However, your tutor will give you verbal feedback on what you have done so far and help if there is a problem that you cannot fix, so make the most of the opportunity.

Also, it is **strongly recommended** that you upload your interim deliverables to Moodle at the time you present them.

Final Deliverable

The following must be uploaded, via the Moodle page for the course, by 23:30 on the deadline date. NOTE THAT THE SERVER IS OFTEN BUSY CLOSE TO THE DEADLINE AND ANY UPLOADS AFTER 22:00 ARE AT YOUR OWN RISK.

A **zip file** containing:

- a PyCharm project
- the project report as a pdf file

The **project report** should contain evidence of all completed stages, including the following **sections**:

- **Introduction**
- **Design and Development:** a description of how you designed and developed the final code with suitable screenshots of the program in operation.
- **Testing and Faults**: including a summary of the test table (the actual table and results will be listed in appendix B) and a discussion of any faults and failures, including those that you managed to correct, and those which are still unresolved.
- **Conclusions, further development and reflection**: Give a summary of the program and discuss what you would do if you had another three months to work on the program. For the reflection you should write around 400 words, answering either (a) or (b) from the following:
    a) What did I achieve with this element of learning? Which were the most difficult parts, and why were they difficult for me? Which were the most straightforward parts, and why did I find these easy?
    b) What have I got out of doing this coursework? How have I developed my knowledge and skills? How do I see this coursework helping me in the longer term?
- Appendices should contain:
    A. The commented version of the code that you worked on for Stage 1.
    B. Test table and results (updated with any changes you have made to the code in Stage 5).

The written part of the report (excluding appendices) should be no more than 2,000 words.

To demonstrate the finished code you should include **up to 10** screenshots demonstrating the key functionality (you do not need to show screenshots of every single operation).

Grading and Feedback

Indicative Grading Criteria

70-100% : A well-designed and implemented program, together with an excellent accompanying report, which shows a very good understanding of programming concepts and demonstrates some imaginative uses of programming techniques. To gain a mark in this range you must have made:

- a very good attempt at all stages, with an excellent report.

60-69% : A well-designed and implemented program, together with a good accompanying report, which shows a good understanding of programming concepts and demonstrates typical uses of programming techniques. To gain a mark in this range you must have made:

- a good attempt at all stages, with a good report; OR
- a very good attempt at all stages, but the report is weak; OR
- a very good attempt at all stages, but the program is hard to use; OR
- a very good attempt at all stages, but the code contains minor runtime errors

40-59% : A reasonably well-designed and implemented program, together with a reasonable accompanying report, which shows some understanding of programming concepts and demonstrates some uses of programming techniques. To gain a mark in this range you must have made:

- a good attempt at stages 1-4, with a good report; OR
- a reasonable attempt at all stages, but the report is missing a section; OR
- a reasonable attempt at all stages, but the code contains serious runtime errors

20-39% : A poorly implemented program, together with an accompanying report, which shows a basic understanding of programming concepts and demonstrates limited uses of programming techniques. The code may contain some errors or the report may be missing a section. To gain a mark in this range you must have made:

- an attempt at stages 1-3, with a reasonable report; OR
- a reasonable attempt at further stages, but the report is missing several sections; OR
- a reasonable attempt at further stages, but the code cannot be run by your tutor

10-19% : A poorly implemented program, together with an accompanying report, which barely shows any understanding of programming concepts and/or only demonstrates very basic uses of programming techniques. To gain a mark in this range you must have made:

- an attempt at stages 1-2

0-9% : A virtually non-existent program and report. To gain a mark in this range you must have made:

- an attempt at stage 1

Feedback

Your work will be graded using the following feedback sheet.

COMP1752 Object Oriented Programming - grading and feedback sheet

This sheet is used to grade your work and give feedback on your coursework.

The first table indicates which **Grading Band** your coursework falls into and why. Marks are then awarded for each stage and your overall mark is based the **Assessment Criteria**, possibly capped by the stage you have reached. You may receive a mark in a higher grading band if your tutor feels some of the work justifies it.

Note: you cannot get marks for a particular stage unless you have made a reasonable attempt at all the previous stages. For example, if you miss out the testing (stage 4) you cannot get a mark over 50%, regardless of how good your innovations are.

Indicative Grading Band & Criteria	Your work
70-100%: a very good attempt at all stages, with an excellent report	
60-69%: a good attempt at all stages, with a good report 60-69%: a very good attempt at all stages, but the report is weak 60-69%: a very good attempt at all stages, but the program is hard to use 60-69%: a very good attempt at all stages, but the code contains minor runtime errors	
40-59%: a good attempt at stages 1-4, with a good report 40-59%: a very good attempt at stages 1-3, with a good report 40-59%: a reasonable attempt at all stages, but the report is missing a section 40-59%: a reasonable attempt at all stages, but the code contains serious runtime errors	
20-39%: an attempt at stages 1-3, with a reasonable report 20-39%: a reasonable attempt at further stages, but the report is missing several sections 20-39%: a reasonable attempt at further stages, but the code cannot be run by your tutor	
10-19%: an attempt at stages 1-2	
0-9%: an attempt at stage 1	

Assessment Criteria	Marks
Stage 1. Basic understanding (10%)	
Commented code for Stage 1 [10 marks]	
Stage 2. Outline design and implementation (10%)	
Design – is the system straightforward and easy to use? [10 marks]	
Stage 3. Basic working version (30%)	
Functionality – does the system do what it is supposed to? [10 marks]	
Clear and concise design documentation [10 marks]	
Well-structured code with appropriate naming & helpful comments [10 marks]	
Stage 4. Testing and validation (20%)	
Evidence of appropriate testing [10 marks]	
Bad input data handled appropriately (validation) [5 marks]	
Unit Testing [5 marks]	
Stage 5. Innovations and future development (30%)	
Discussion & implementation of innovation 1 [10 marks]	
Discussion & implementation of innovation 2 [10 marks]	
Reflection on the learning process and future developments [10 marks]	

Overall Mark:

Feedback and Feedforward for next assignment
What you did well in this assignment:
What you could improve in this assignment:
What you can take forward to your next assignment: